**Ternary Search and Interpolation Search Algorithms**

**1. Ternary Search**

Ternary search is a divide-and-conquer search algorithm that works on sorted arrays. It divides the array into three parts and determines which part may contain the desired element. Here's how it works and its complexity analysis:

**How Ternary Search Works:**

**Step 1:** Choose two middle elements, generally called mid1 and mid2. These are located at positions:

* mid1 = low + (high - low) / 3
* mid2 = high - (high - low) / 3

**Step 2:** Compare the target value (key) with arr[mid1] and arr[mid2]:

* If key == arr[mid1], return mid1.
* If key == arr[mid2], return mid2.
* If key < arr[mid1], perform the search in the left third of the array.
* If key > arr[mid2], perform the search in the right third of the array.
* Otherwise, search in the middle third.

**Step 3:** Repeat the process with updated boundaries (low and high).

**Complexity Analysis:**

**Time Complexity**: Since we are dividing the search space into three parts at each step, the time complexity of ternary search is:

* Best, Average, and Worst Case: O(log₃ N)

**Space Complexity:** The space complexity is O(1) because it uses constant extra space.

**Ternary Search C++ Implementation:**

#include <iostream>

using namespace std;

int ternarySearch(int arr[], int low, int high, int key) {

if (high >= low) {

int mid1 = low + (high - low) / 3;

int mid2 = high - (high - low) / 3;

if (arr[mid1] == key)

return mid1;

if (arr[mid2] == key)

return mid2;

if (key < arr[mid1])

return ternarySearch(arr, low, mid1 - 1, key);

else if (key > arr[mid2])

return ternarySearch(arr, mid2 + 1, high, key);

else

return ternarySearch(arr, mid1 + 1, mid2 - 1, key);

}

return -1;

}

int main() {

int arr[] = {1, 5, 9, 12, 16, 20, 24, 28, 32, 35};

int n = sizeof(arr) / sizeof(arr[0]);

int key = 16;

int result = ternarySearch(arr, 0, n - 1, key);

if (result != -1)

cout << "Element found at index " << result << endl;

else

cout << "Element not found" << endl;

return 0;

}

**2. Interpolation Search**

Interpolation search is an improvement over binary search for uniformly distributed data. It assumes that the values in the array are uniformly distributed and that the key is more likely to be near the value at the middle of the array.

**How Interpolation Search Works:**

**Step 1:** Calculate the estimated position of the target element using the formula:

pos = low + [(key - arr[low]) \* (high - low)] / (arr[high] - arr[low])

This formula estimates the position of the key based on the assumption that the elements are uniformly distributed.

**Step 2:** Compare the key with the element at the estimated position:

* If arr[pos] == key, return pos.
* If arr[pos] < key, move the search to the right half of the array (low = pos + 1).
* If arr[pos] > key, move the search to the left half (high = pos - 1).

**Step 3:** Repeat the process until the element is found or the search bounds (low > high) are invalid.

**Complexity Analysis:**

**Time Complexity:**

* Best Case: O(log log N), when the distribution is nearly uniform and the key is located near the middle.
* Worst Case: O(N), when the distribution is skewed or if the values are not uniformly distributed.
* Average Case: O(log log N), under the assumption of a uniform distribution of values.

**Space Complexity:** O(1), as it uses constant space.

**Interpolation Search C++ Implementation:**

#include <iostream>

using namespace std;

int interpolationSearch(int arr[], int low, int high, int key) {

while (low <= high && key >= arr[low] && key <= arr[high]) {

// Estimate position using interpolation formula

int pos = low + ((key - arr[low]) \* (high - low)) / (arr[high] - arr[low]);

// Check if the key is at the position

if (arr[pos] == key)

return pos;

// If key is larger, move to the right

if (arr[pos] < key)

low = pos + 1;

// If key is smaller, move to the left

else

high = pos - 1;

}

return -1;

}

int main() {

int arr[] = {10, 12, 20, 32, 35, 40, 45, 50};

int n = sizeof(arr) / sizeof(arr[0]);

int key = 35;

int result = interpolationSearch(arr, 0, n - 1, key);

if (result != -1)

cout << "Element found at index " << result << endl;

else

cout << "Element not found" << endl;

return 0;

}

**Summary**

**Ternary Search:**

* Works by dividing the search range into three parts.
* Best for sorted arrays.
* Time Complexity: O(log₃ N), Space Complexity: O(1).

**Interpolation Search:**

* More efficient for uniformly distributed data by estimating the position of the key.
* Best when the data is uniformly distributed.
* Time Complexity: O(log log N) on average, O(N) in the worst case, Space Complexity: O(1).

Both algorithms offer improvements over simple linear search for specific conditions, but they require a sorted array to operate efficiently.